1 たとえ話

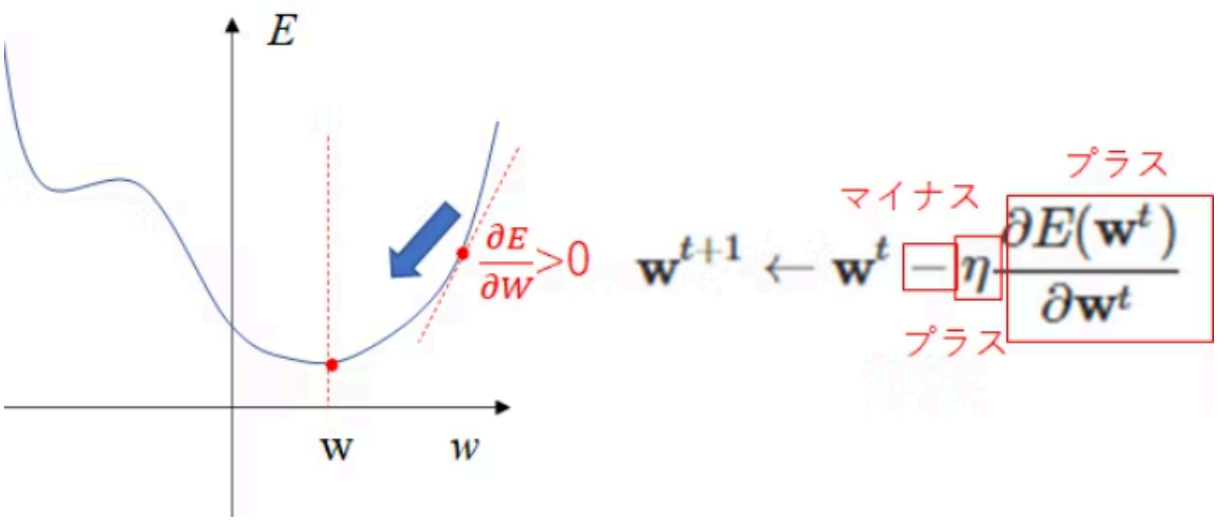
地図も見ずに目隠しで谷底を探すという意味不明な例だったので、、、

自分を山の頂上にいる登山家だと想像してください。夜が訪れてしまいました。麓にあるベースキャンプに戻る必要がありますが、暗い中で貧弱な懐中電灯しか持っておらず、目の前にある 1 メートル程度先の地面しか見ることができません。この際にどのように下りるのが一番良いのでしょう。

戦略の 1 つは、あらゆる方向を見てどの道が最も急に下っているのかを見極め、その方向に進むことです。このプロセスを何回も繰り返すと、徐々に山を下りることができるでしょう。たまに小さな凹みや谷に嵌るかもしれません。その場合は、勢いをつけてもう少し進むことで抜け出すことができます。細かな欠点は置いておくとして、この戦略に従えば最終的には麓まで辿り着くことができます。[1]

2 確率的勾配降下法 SGD

$$w^{t+1} \leftarrow w^t - \eta \frac{\partial E(w^t)}{\partial w^t} \tag{1}$$



ηは固定値（0.01 とか 0.001 とかを取ります）となっているため、モデルを作る人間が手動で値を決める[2]必要があります。勾配の値で最小値に向かっていくため、損失関数によってはハンチングをしたりすることで、収束しないことがある。
機械学習をさせたい人間としては、この値をなるべく自動で最適化させたいと思うはず！より早く収束に向かわせるように調整したアルゴリズムが次の

3 モーメンタム Momentum

モーメンタム法は勾配降下法のうち、重みの更新に慣性を用いる種類を指します。言い換えると、重みはもはや現在の時刻における勾配だけの関数ではなく、過去の更新レートを元に徐々に調整される。物理学の力学分野でいうところの慣性と同じ考え方をすることから、Momentum(慣性項）と呼ばれます。 α はこの慣性項のパラメータになる。

[1] https://ml4a.github.io/ml4a/jp/how_neural_networks_are_trained/
[2] このように、手動で調整するため自動的に値が最適化されない値を**ハイパーパラメータ**と呼びます。

$$v^{t+1} \leftarrow \alpha v^t - \eta \frac{\partial E(w^t)}{\partial w^t} \tag{2}$$

$$w^{t+1} \leftarrow w^t - v^{t+1} \tag{3}$$

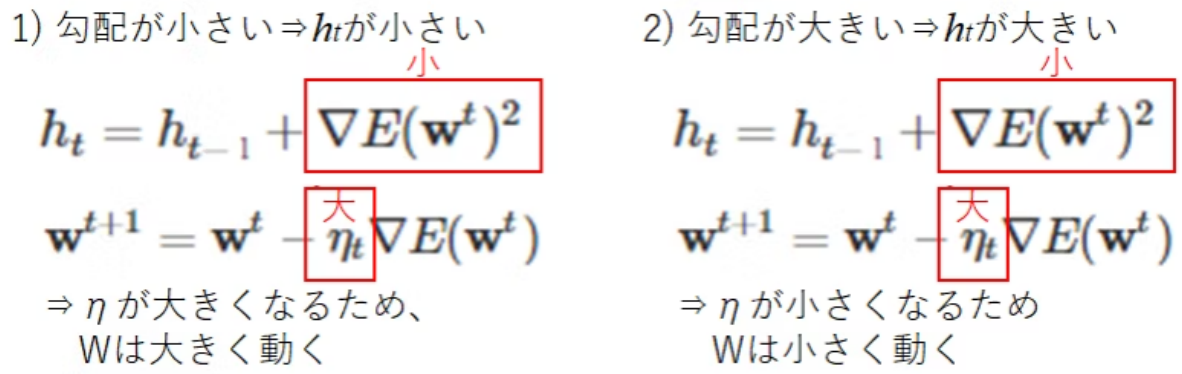
更新の道筋は坂を下るボールのように考えることができます。勾配が大きく変化する場所にボールがさしかかっても、勢いがあるため勾配に沿って少しずつ向きを変えるだけでほぼ同じ方向に進み続けます。モーメンタム法は過去の更新から積み重ねたスピードに乗って鞍点(（あんてん)[3]や極小値から抜け出す助けにもなります。また、損失関数の曲面が局地的に一定の向きにだけ大きく傾いている場所ではジグザクな蛇行がしばしば問題になりますが、勢いがあることでこの問題にも対抗できます。 しかしながら、この場合も学習率 η や α はある値を取っている定数であるため、最適化が自動で行えない課題が残っている。

4 適応的勾配 AdaGrad (アダグラッド?)

学習率を勾配によって変わる変数としています。勾配の 2 乗となっています。従って必ず正の値を取ります。その値が加算されていくため、それを分母に取る学習率は徐々に低い値を示していきます。これは、最適点に落ち着かない現象を回避してくれる効果があります。

$$h^{t+1} \leftarrow h^t + \left(\frac{\partial E(w^t)}{\partial w^t} \right)^2 \tag{4}$$

$$w^{t+1} \leftarrow w^t - \eta \frac{1}{\sqrt{h}} \frac{\partial E(w^t)}{\partial w^t} \tag{5}$$



- よく変わる数字はゆっくり、あまり変わらない数字は速く: 頻繁に更新される数字はゆっくり学習し、あまり更新されない数字は速く学習します。

学習率が変化するようになったことで落ち着いて最適点に収束していきそうな気がします。しかしここでも課題が出てきます。学習回数（epochs）を重ねていくと、更新量が 0 に近づいて行ってしまうため更新がされなくなってしまいます。最適点に落ち着いていない場合はこれでは問題です。

5 Adam

Adam を解説するまえに、RMSProp から説明します。

[3]多変数実関数の変域の中で、ある方向で見れば極大値だが別の方向で見れば極小値となる点

5.1 RMSProp

AdaGrad のhについてみると、過去全部を用いるわけではなく、直近の勾配の二乗和をとるようになっています。α は 0.99 などの値を指定します。すると下記 に関する式について、第 2 項の値が小さい一方で第 1 項の方が値が大きくなることが分かります。

$$h^{t+1} \leftarrow \alpha h^t + (1-\alpha)\left(\frac{\partial E(w^t)}{\partial w^t}\right)^2 \tag{6}$$

$$w^{t+1} \leftarrow w^t - \eta \frac{1}{\sqrt{h}} \frac{\partial E(w^t)}{\partial w^t} \tag{7}$$

つまり AdaGrad よりも α の値を調整することで直近とそれより以前の勾配の影響どちらを考慮させたいかを適切に選んで最適化に向かわせることができます。

5.2 Adam

Adam とは、Adaptive（適応性のある）、Moment(運動量)、Estimation（見積）の略になります。

考え方としては、Momentum で出てきた物理法則に準じる動きを取り入れることと、AdaGrad で出てきた学習率を適宜変化させていくことを取り入れたハイブリッドなアルゴリズムになります。

$$m^{t+1} \leftarrow \beta_1 m^t + (1-\beta_1)\frac{\partial E(w^t)}{\partial w^t} \tag{8}$$

$$v^{t+1} \leftarrow \beta_2 v^t + (1-\beta_2)\left(\frac{\partial E(w^t)}{\partial w^t}\right)^2 \tag{9}$$

$$w^{t+1} \leftarrow w^t - \eta \frac{m^{t+1}}{\sqrt{v^{t+1}}} \tag{10}$$

m^{t+1}やv^{t+1}はバイアス補正と呼ばれる値となっています。これは、ある定数によって割った値を示しています。通常、β_1は 0.9、β_2は 0.999 を示します。

AdaGrad や RMSProp は学習率だけに対して調整を行うようなものを Adam は勾配の 2 乗平均と平均を 1 次モーメントと 2 次モーメントとして考慮することで、パラメータごとに適切なスケールで重みが更新されることを可能にしました。[4]

common/optimizar.py からのコード

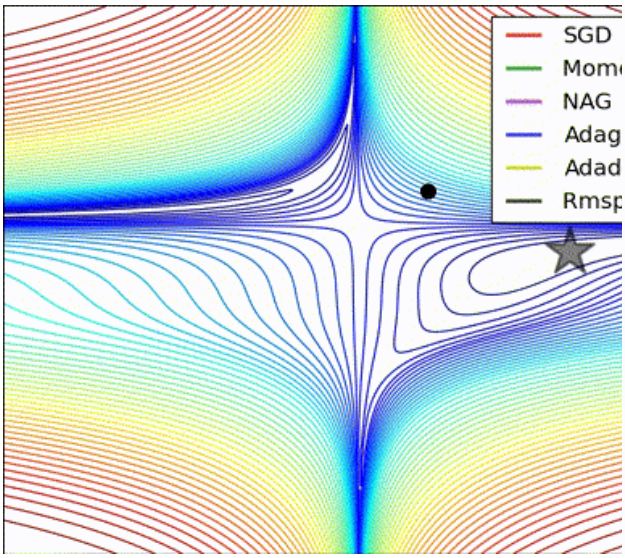
```
class Adam:
    # __init__(lr, beta1, beta2)
    def __init__(self, lr=0.001, beta1=0.9, beta2=0.999):
        self.lr = lr  # 学習率
        self.beta1 = beta1  # 一次モーメントの減衰率
        self.beta2 = beta2  # 二次モーメントの減衰率
        self.iter = 0  # イテレーション回数
        self.m = None  # 一次モーメント
        self.v = None  # 二次モーメント

    # update(params, grads)
```

[4] より詳しく Adam について->https://qiita.com/exp/items/99145796a87cc6cd47e1

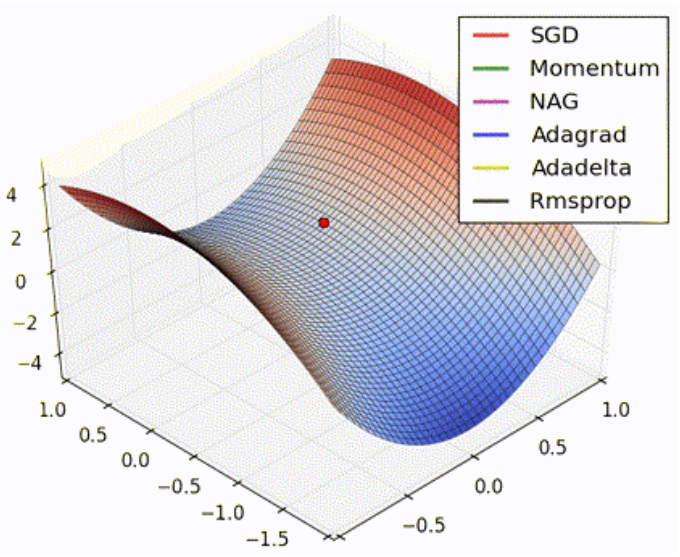
```
    def update(self, params, grads):
        # 初回呼び出し時の初期化
        if self.m is None:
            self.m, self.v = {}, {}
            for key, val in params.items():
                self.m[key] = np.zeros_like(val)
                self.v[key] = np.zeros_like(val)

        self.iter += 1
        # 学習率の補正
        lr_t  = self.lr * np.sqrt(1.0 - self.beta2**self.iter) / (1.0 -
self.beta1**self.iter)

        for key in params.keys():
            # 一次モーメントの更新
            self.m[key] += (1 - self.beta1) * (grads[key] - self.m[key])
            # 二次モーメントの更新
            self.v[key] += (1 - self.beta2) * (grads[key]**2 - self.v[key])

            # パラメータの更新 1e-7 は 0 除算防止の定数
            params[key] -= lr_t * self.m[key] / (np.sqrt(self.v[key]) + 1e-7)
```

6 更新方法の比較

モーメンタム法とネステロフ加速勾配降下法（NAG）は「下り坂を転がる」速度が付きすぎて最適な経路をオーバーシュートする傾向があることに対し、標準的な SGD は適切な経路を取るものの遅すぎることに注意。適応的手法である AdaGrad、AdaDelta、RMSProp（Adam もここに含まれます）はパラメータごとの柔軟性があることで、これらの問題を回避する傾向があります。



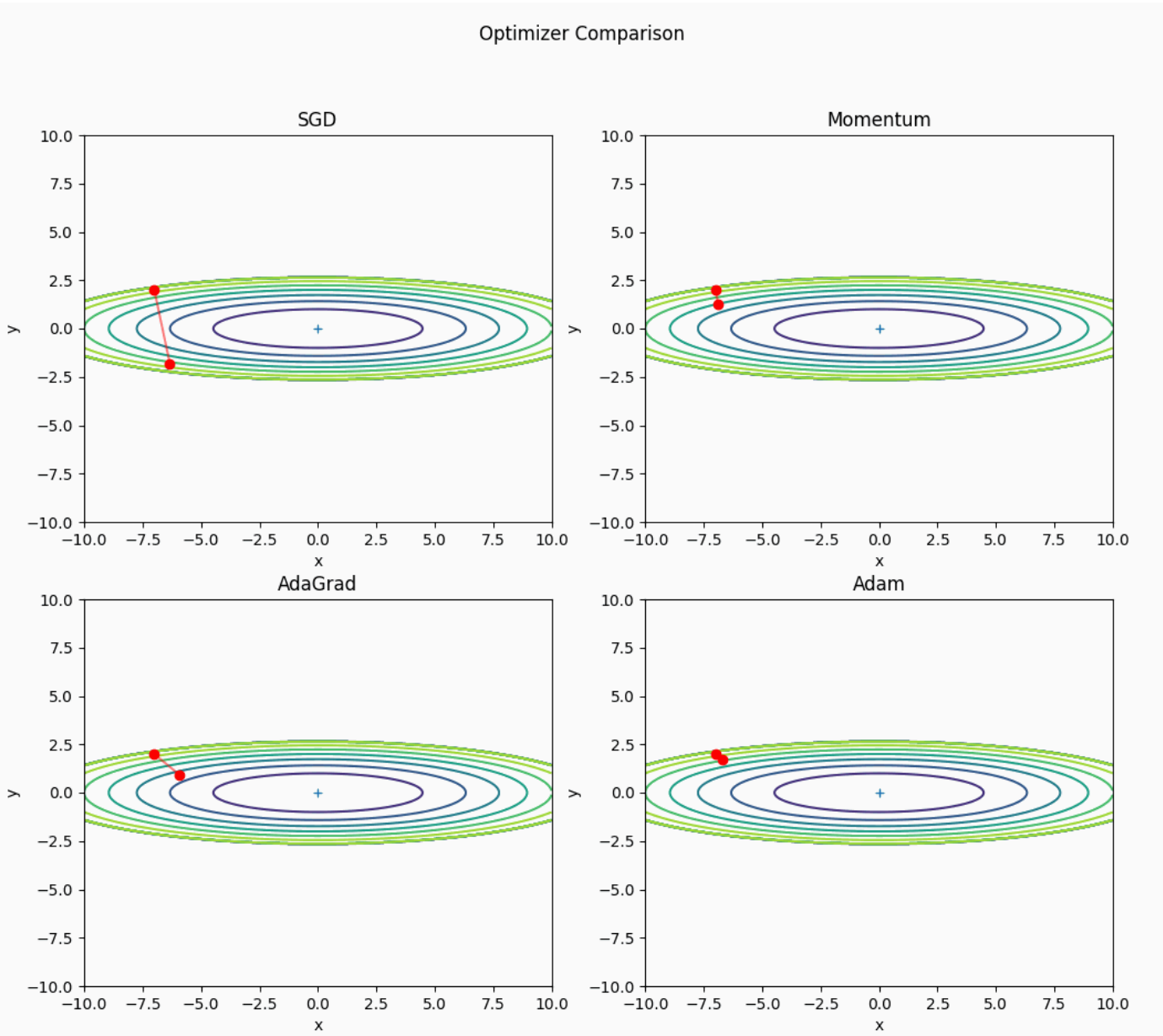
勾配更新手法が目的のパラメータに収束する様子の等高線図。[5]



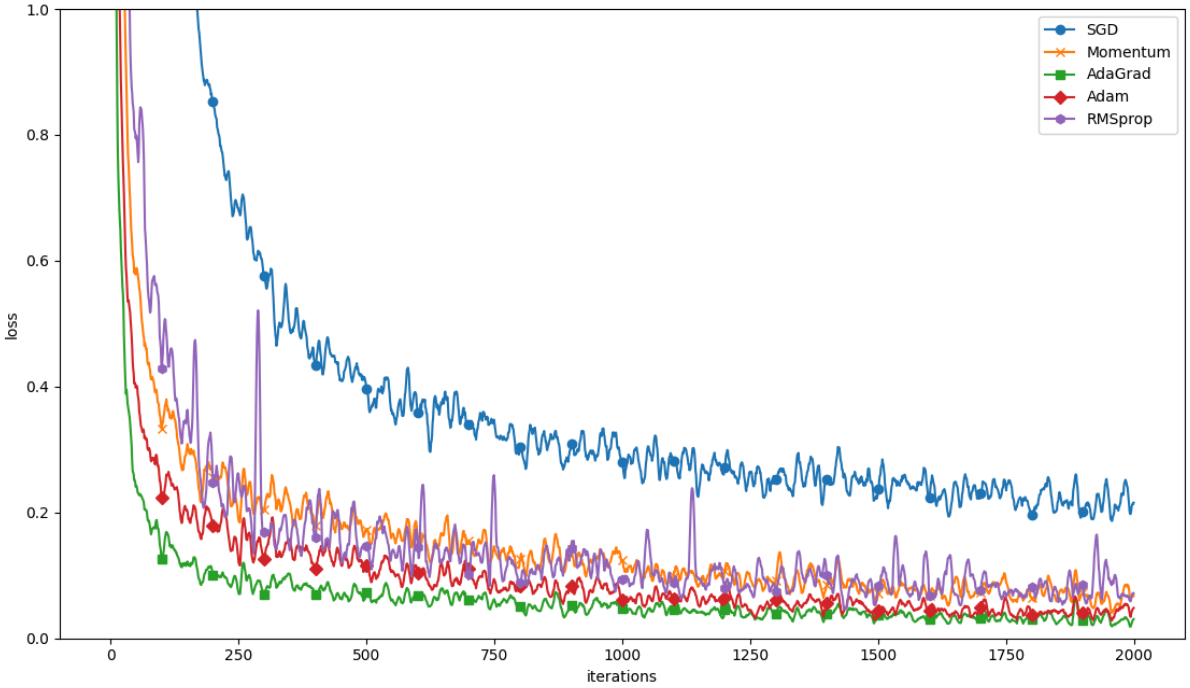
勾配更新手法が鞍点を抜け出す様子の比較。SGD が鞍点にはまってしまうことに注意。[6]

鞍点（あんてん、英 saddle point）: 多変数実関数の変域の中で、ある方向で見れば極大値だが別の方向で見れば極小値となる点である。

[5] https://www.twitter.com/alecrad
[6] https://www.twitter.com/alecrad



赤点は計算速度に沿っていないので注意



ch06/optimizer_compare_mnist.py から、実行時間がかかるため結果のみ載せます。また、RMSprop の結果も乗せました。上図より sgd が少し遅いとわかる。

7 重みの初期値と過学習の関係

過学習 モデルがトレーニングセットを正確に予測するように最適化され過ぎてしまい、（学習の本来の目的である）未知のデータに対応する汎用性が失われてしまっている状態のことを指します。

これはモデルが、トレーニングセットに含まれるノイズまで拾って、データに完全に沿うように大きくねじ曲がってしまった場合にも起こります。

“過学習とは、アルゴリズムがある種のインチキをしているのだと考えることもできます。知っているデータに対してだけ誤差が最小限になるようにして、見せ掛けだけの高得点を出せるとあなたを信じ込ませようとしているのです。これはファッションの仕組みを学ぼうとしているのに、70 年代のディスコにいる人々の写真しか見たことがなくて、ベルボトム、デニムジャケット、厚底の靴が全てだと思っているようなものです。親しい友達や家族にもそんな人がいるかもしれませんね。”[7]

[7]https://ml4a.github.io/ml4a/jp/how_neural_networks_are_trained/

右の画像は黒い点で示された 11 個のサンプルに対して適合するように、2 つの関数を訓練します。1 つは直線で、大まかにデータの特徴を捉えています。もう 1 つはとても曲がりくねった線で、全てのサンプルに完璧に一致しています。一見、後者の方が誤差が少ない（実際に 0 です）ので、トレーニングデータに対して良く適合しているように思えるかもしれません。しかしそれは潜在的なばらつきをうまく捉えることができておらず、未知のデータに対しては残念な性能しか出せないでしょう。

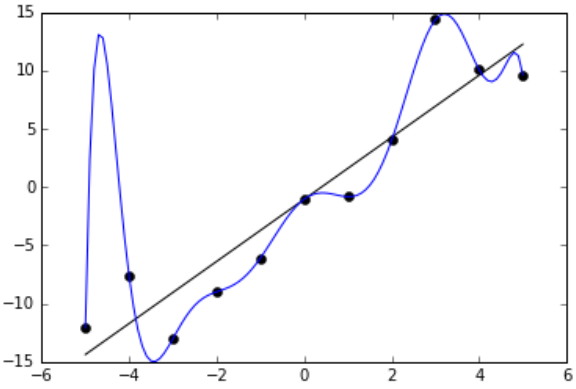


Figure 2: 直線はシンプルで多少の誤差はあるものの、大まかにデータを捉えている。曲がりくねった線は誤差 0 だが非常に複雑で、おそらくうまく一般化ができてない。[8]

重みの初期値を小さくすることで過学習を抑える効果があります。これには主に 2 つの理由があります。

- モデルの複雑さの制御 重みが大きいと、ニューラルネットワークの各層で入力値が大きく変換されます。これにより、モデルが複雑になりすぎて、訓練データの細かな特徴まで学習してしまう可能性が高くなります。
  例:
  - 小さな重み: 入力 1 × 重み 0.1 = 出力 0.1
  - 大きな重み: 入力 1 × 重み 10 = 出力 10

  大きな重みの場合、小さな入力の変化でも出力に大きな影響を与えてしまいます。
- 勾配消失問題の回避

重みが大きすぎると、特にシグモイド関数などの活性化関数を使用する場合に勾配消失問題が発生しやすくなります。\

勾配消失問題 活性化関数の出力が飽和状態（0 または 1 に近い値）になると、勾配が非常に小さくなり、学習が進まなくなる現象です。

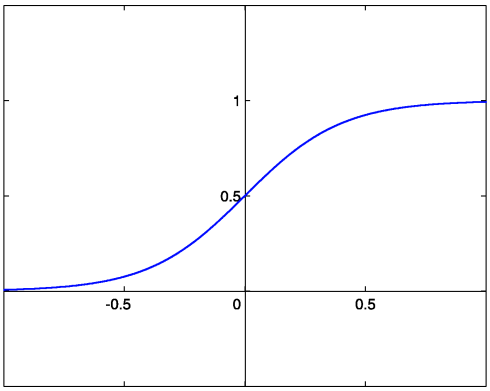
シグモイド関数も、逆伝播の微分値は $y(1-y)$ なので、重みが大きいと y も大きくなって微分値が小さくなり、逆伝播でかけ合わさっていくと勾配がどんどん小さくなって学習が進まなくなる。

$$\boldsymbol{y = \frac{1}{1 + \exp(-x)}} \tag{11}$$

確率的勾配降下法 SGD より、 $y(1-y)$は$\frac{\partial E(w^t)}{\partial w^t}$なので、勾配が小さくなるため、$w^{t+1}$の更新値がどんどん小さくなる。

$$\boldsymbol{w^{t+1} \leftarrow w^t - \eta \frac{\partial E(w^t)}{\partial w^t}} \tag{12}$$

[8] https://en.wikipedia.org/wiki/Overfitting



(11)

対策:ReLU 関数の利用 正の入力に対して勾配が常に 1 となるため、勾配消失が起きにくい

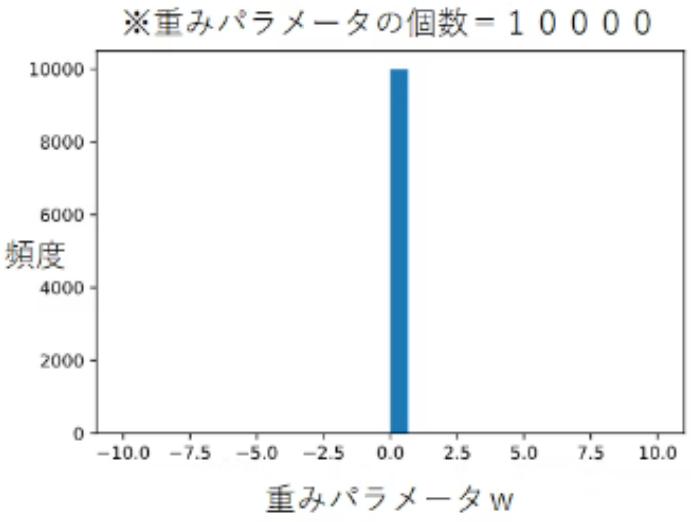
→正の入力に対しては、ReLU の影響で勾配が維持され

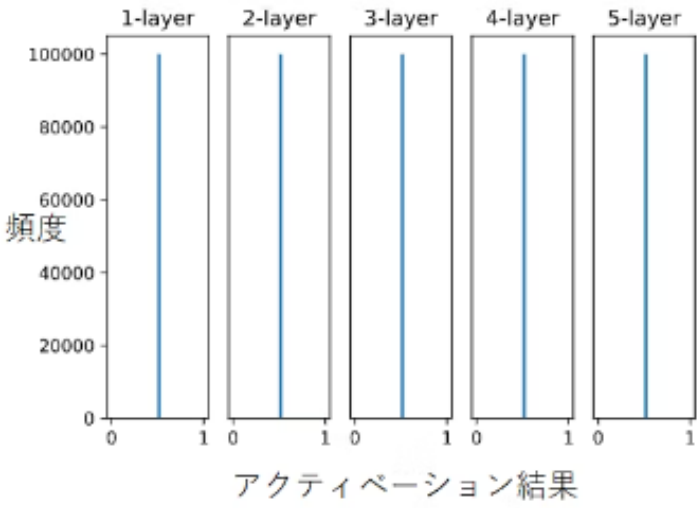
→負の入力に対しては、その部分の勾配が 0 になり、その経路での学習が停止

死んだ ReLU 問題 ニューラルネットワークの学習中に ReLU 関数が「死んで」しまう現象をいいます。[9]

具体的には、ReLU 関数が負の値を持つ入力に対しては常に 0 を返すため、一定のニューロンが学習中に全く活動しなくなる問題です。この状態になると、そのニューロンは重みが更新されなくなり、モデル全体のパフォーマンスが低下します。 (例：ニューラルネットワークが多くの層を持つ場合、ある層のニューロンが死んだ ReLU 問題を起こすと、そのニューロンが下流の層に何も伝えられなくなります。 結果として、モデルが学習を進める能力が著しく制限されます。)

重みパラメータが 0





アクティベーションの値は 0.5 に集中する結果となりました。勾配消失の問題は起きていないが、偏りがあることはモデルの表現力に問題があることを意味します。複数のニューロンから同じ値を出力するとすると、複数存在している意味合いが無くなるからです。つまり層の数を少なくしても同様の結果となるということ。[10]

[9]https://dc-okinawa.com/ailands/unleaky-relu/#toc4
[10]https://qiita.com/Fumio-eisan/items/d697fbd96347ef7e49d5

8 アクティベーション(活性化関数)分布

隠れ層の各々のノードの出力の分布をアクティベーション分布と呼びます。アクティベーション分布が偏るということは、複数のノードが同じ出力をしているということを意味します。

1000 個のランダムなデータサンプルが、100 ニューロンを持つ 5 層のネットワークを通過する様子をシミュレートしている。

ディープラーニングにおけるデータの流れと変換を理解するための例で、実際のニューラルネットワークの学習は行っていませんが、データがネットワークを通過する際の変化を観察できる。

上から、シグモイド(sigmoid)、レルー(ReLU)、タンエイチ(tanh)の順となる。

1000 行 100 列の 2 次元配列:

- サンプル数：1000
- 各サンプルの特徴数：100

標準正規分布（平均 0、標準偏差 1）に従う乱数を生成

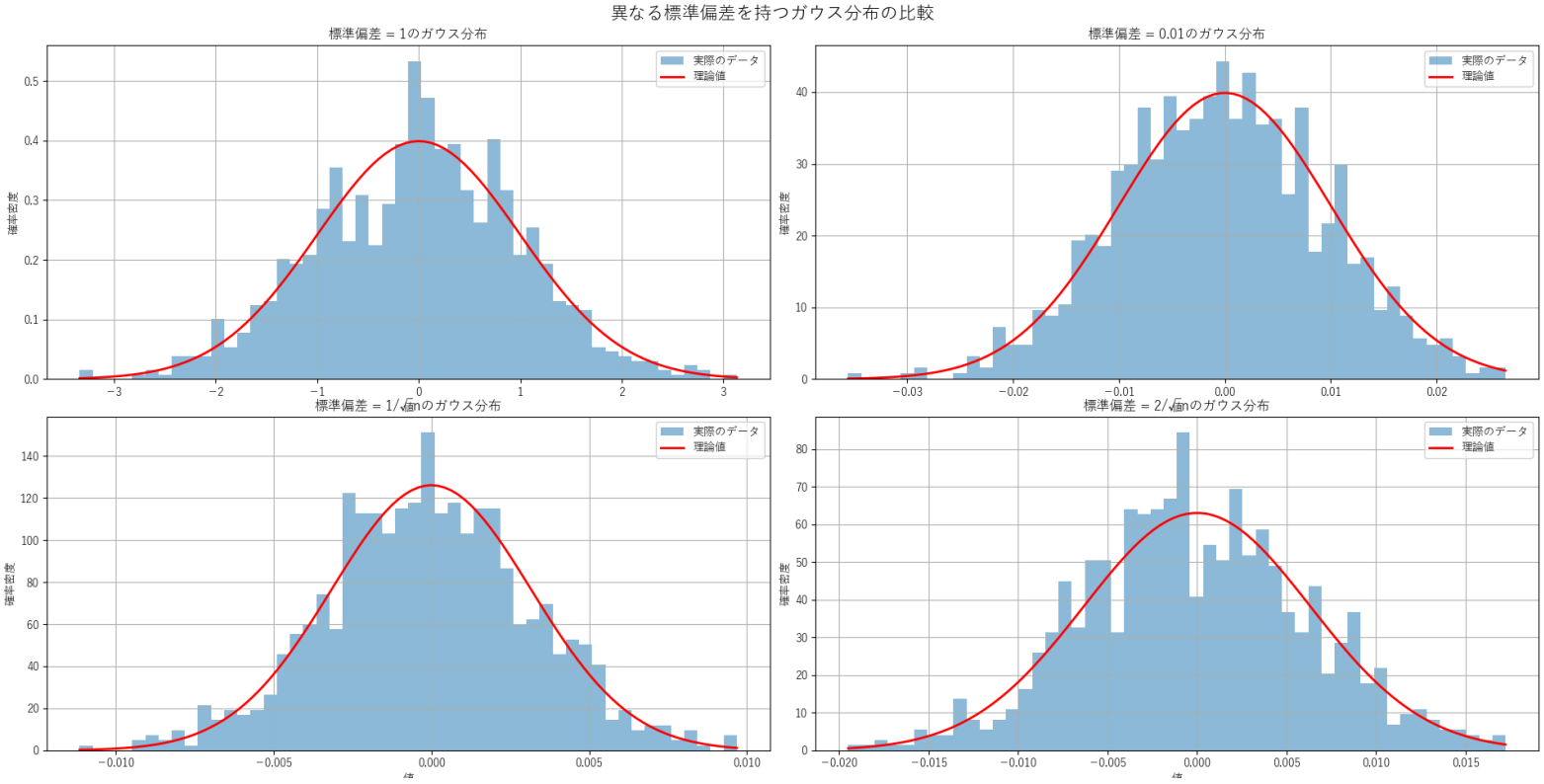
8.0.1 ガウス分布

標準正規分布（平均 0、標準偏差 1):
分布の中心: 平均が 0 なので、データは 0 を中心に分布。

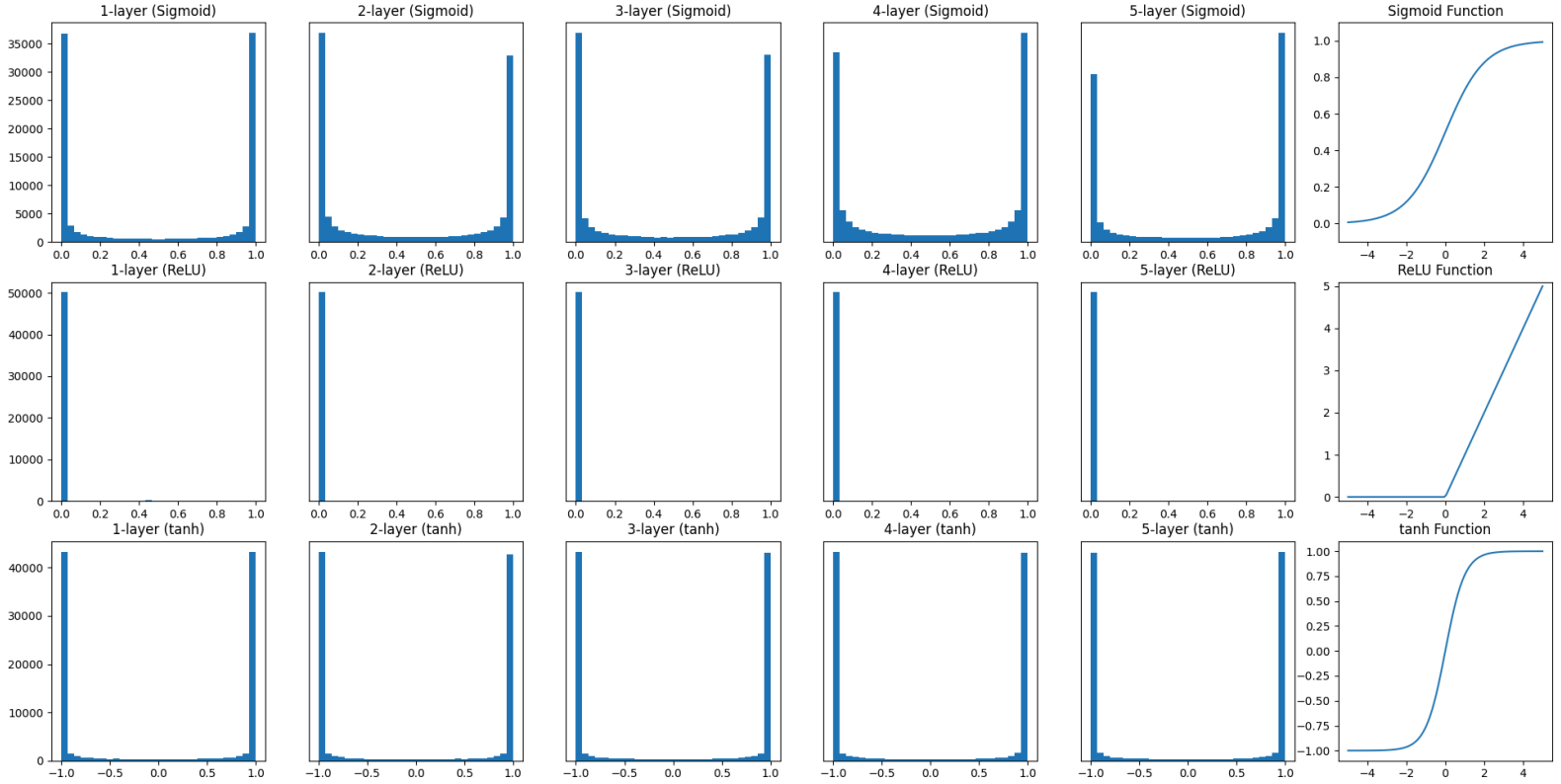
データの範囲:

- 平均値から±標準偏差 1 個分に含まれるデータは全体の約 68%を占める
- 平均値から±標準偏差 2 個分に含まれるデータは全体の約 95%を占める

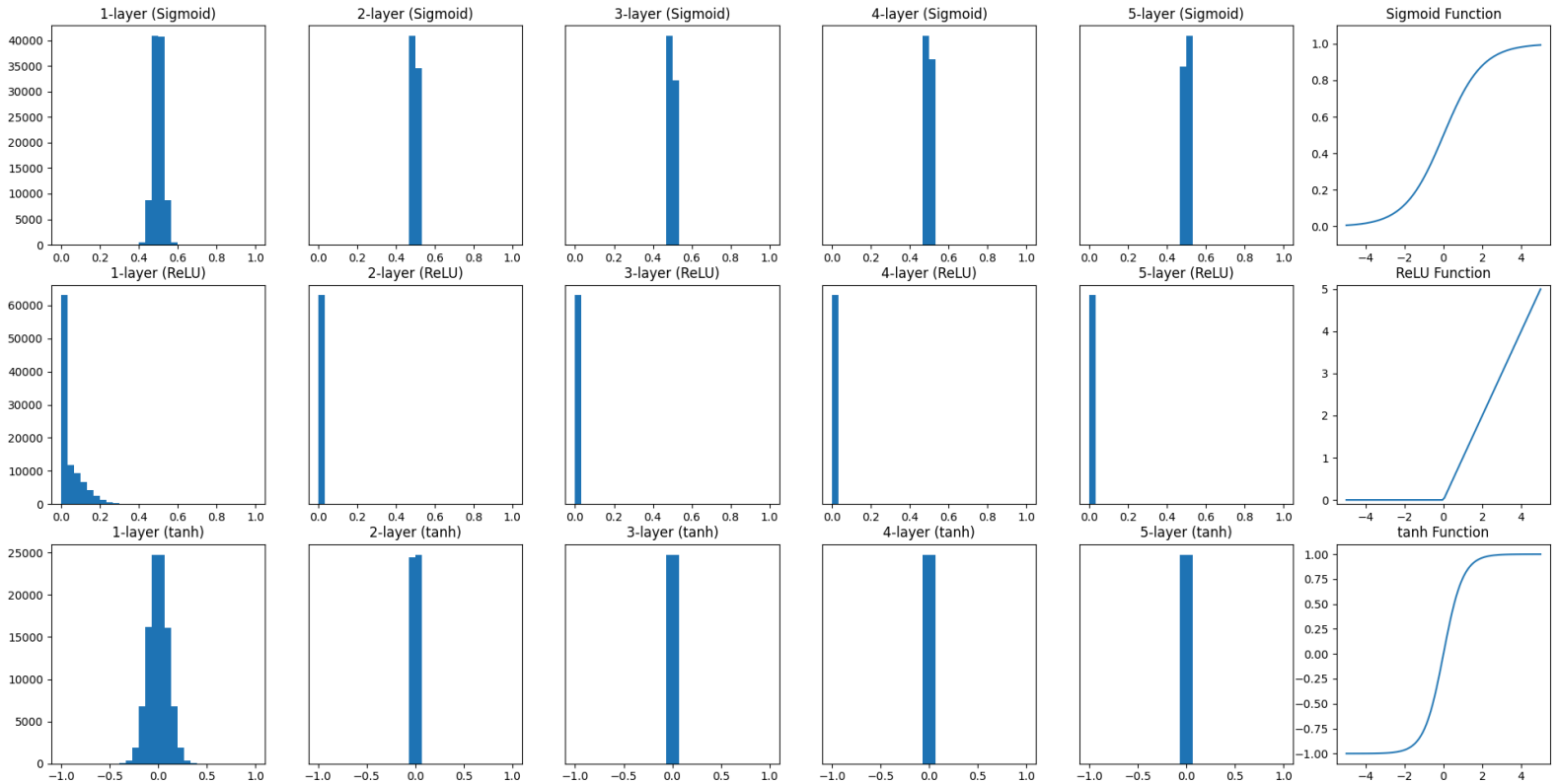
下図の左上は標準偏差 1 、左下は標準偏差$\frac{1}{\sqrt{n}} = \frac{1}{\sqrt{100000}}$約 0.003、右上は 0.01、右下は$\frac{1}{\sqrt{n}} = \frac{2}{\sqrt{100000}}$約 0.006 である。



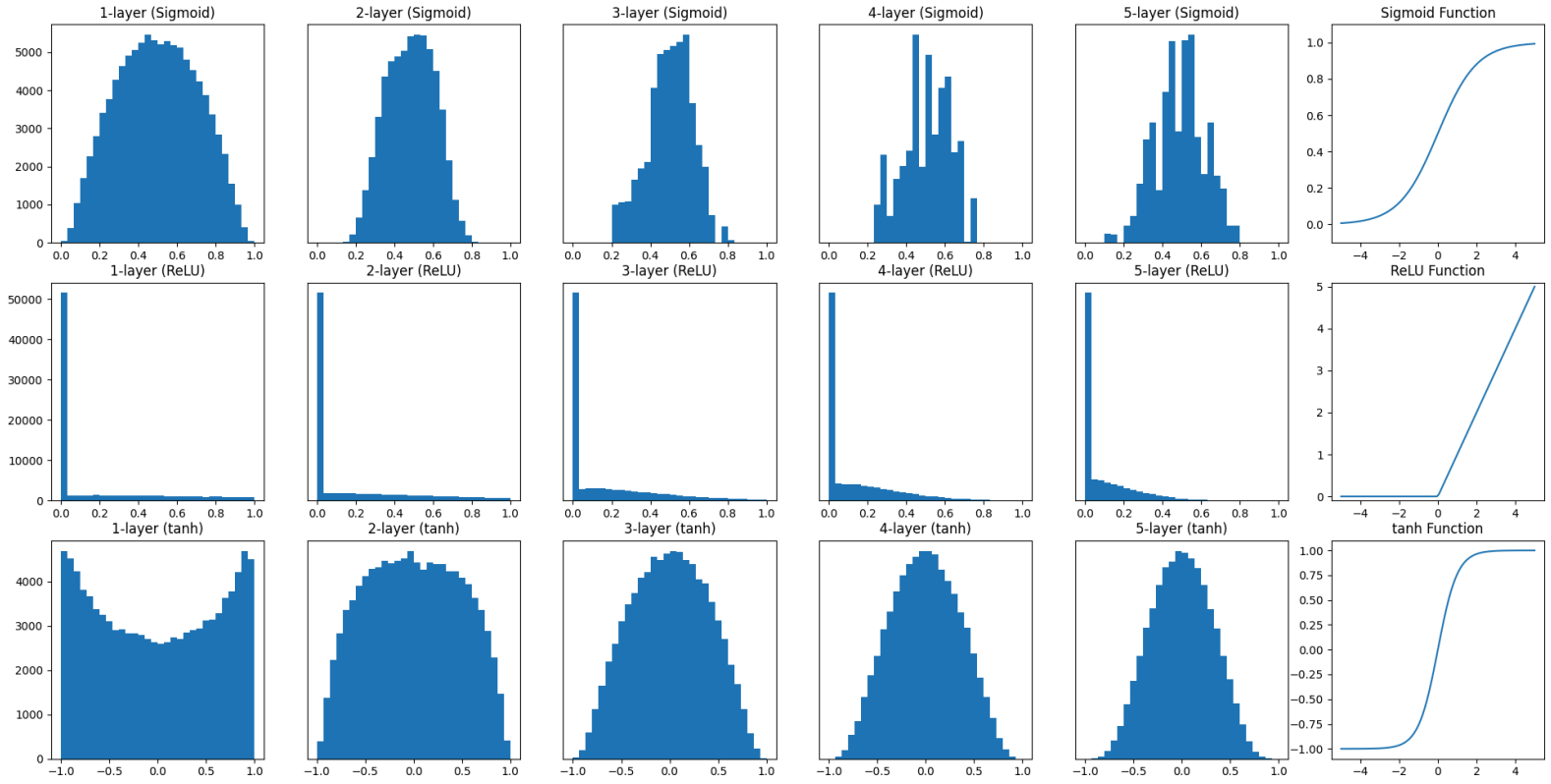
8.0.2 重みの初期値として標準偏差 1 のガウス分布

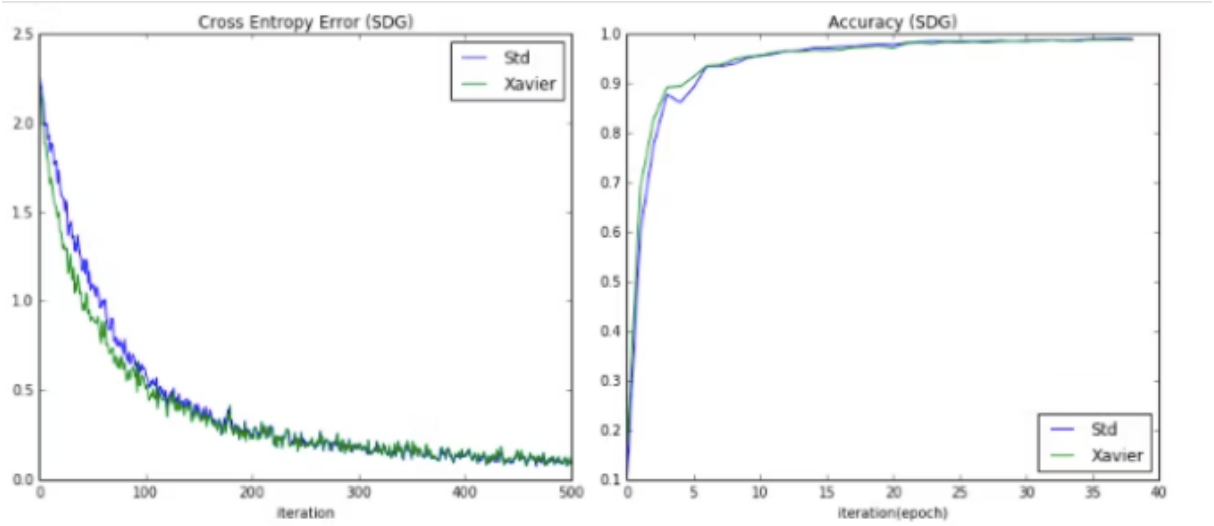


8.0.3 重みの初期値として標準偏差 0.01 のガウス分布



8.0.4 重みの初期値として標準偏差 $\frac{1}{\sqrt{n}}$ のガウス分布 Xavier の初期値->線形に強い Xavier Glorot->ザビエル???

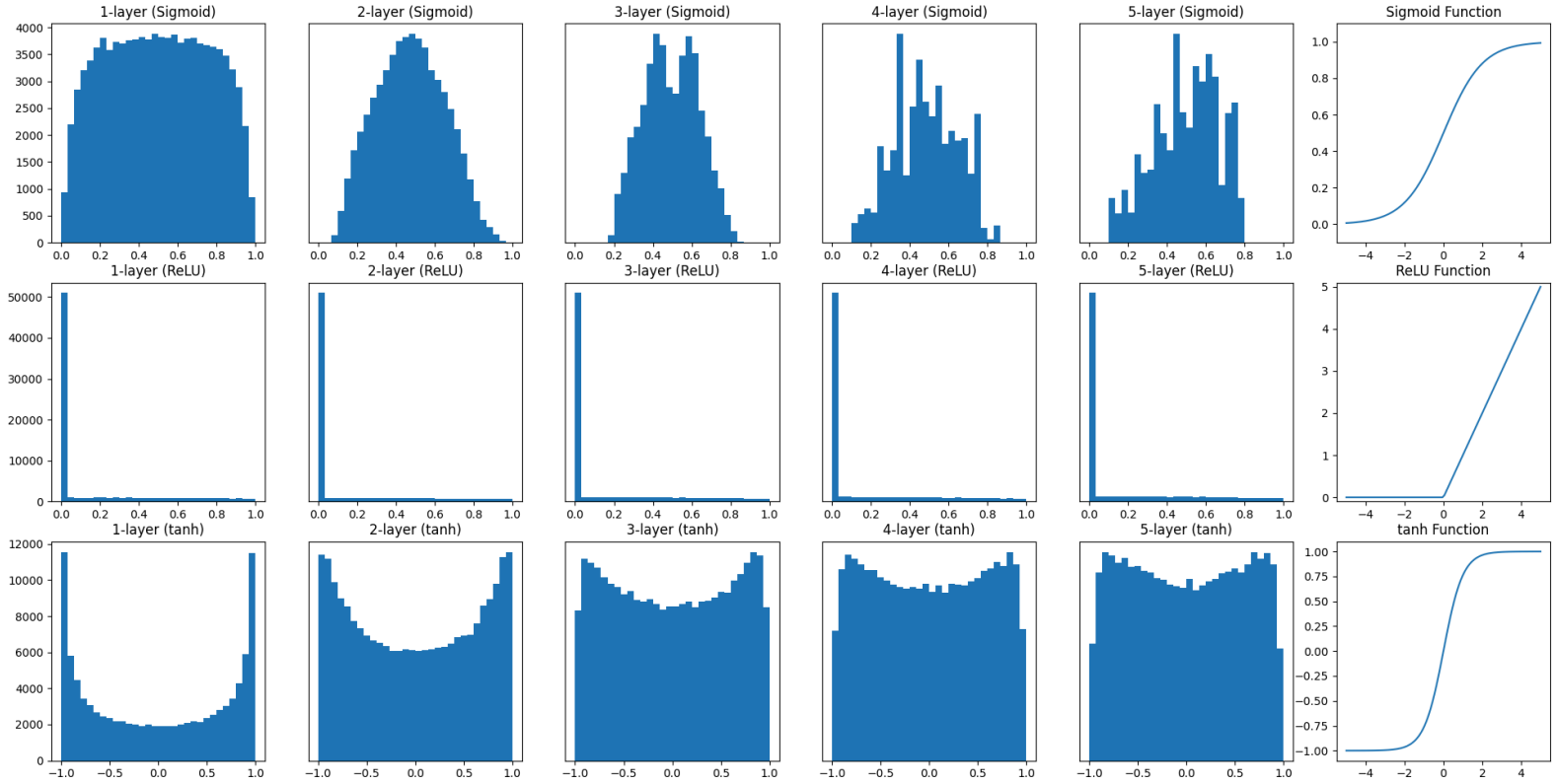


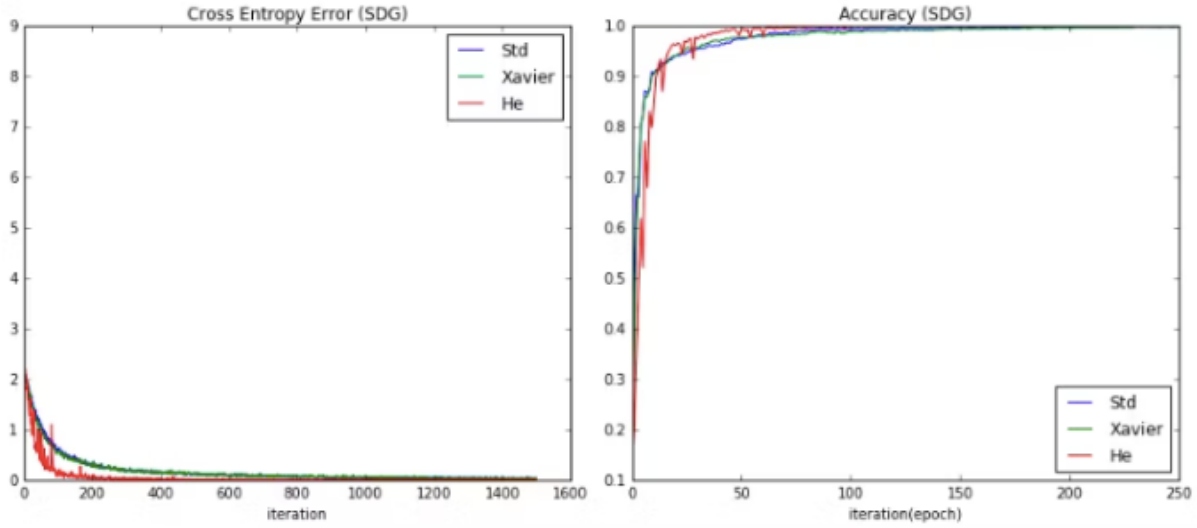


青線が定数 0.1 の初期値重み、緑線が Xavier の初期値です。 ほとんど推移は変わりませんが、若干 Xavier の方が学習推移が早い結果となりました。[11]

[11]https://qiita.com/m-hayashi/items/02065a2e2ec3e2269e0b

8.0.5 重みの初期値として標準偏差 $\frac{2}{\sqrt{n}}$ のガウス分布 He の初期値->ReLU 関数に強い 何恺明 (Kaiming He)->he

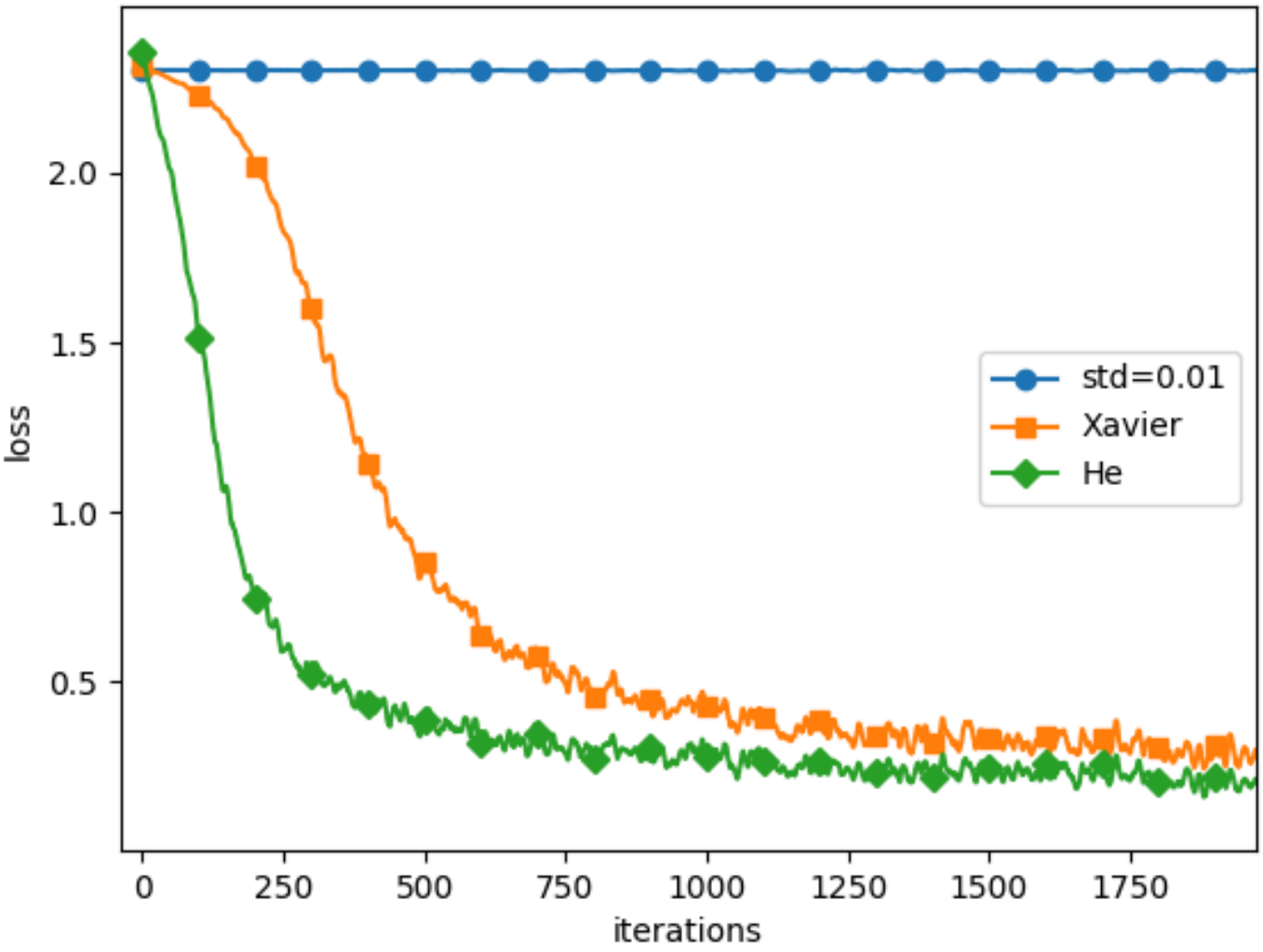




青線の定数 0.1（Std）の重み付けの推移と、緑線の Xavier の初期値での推移は 結果的に大きな差はありませんでしたが、 赤線の He の初期値の採用したものは、交差エントロピー誤差の減衰スピードが 他のものより早い結果が得られました。

ただ、He の場合は活性化関数で ReLU を採用した影響なのか、グラフの推移の振動幅が 大きいという特徴も見られます。[12]

[12]https://qiita.com/m-hayashi/items/02065a2e2ec3e2269e0b



実行時間がかかるため結果のみ載せます。

9 Batch Normalization

重みの初期値の説明では、各層のアクティベーションの分布をいい感じにバラつかせようとしていましたが、Batch Normalization は各層の途中で強制的に分布を正規化してしまおうという手法。

9.1 正規化[13]

正規化（Normalization） データのスケール（単位）を扱いやすいものに整えることである。正規化にはさまざまな方法が考えられるが、主要な方法に、

- 最小値 0〜最大値 1 にスケーリングする「**Min-Max normalization**」
- 平均 0、標準偏差 1 にスケーリングする「**Z-score normalization**」

通常、単に「正規化」と言った場合は、Min-Max normalization を指す。この場合の正規化とは、データの最小値からの偏差（＝最小値を中心 0 にした場合の値）をデータ範囲（＝最

[13] https://atmarkit.itmedia.co.jp/ait/articles/2110/07/news027.html

大値－最小値）で割ることである。これにより、データの最小値は 0、最大値は 1 に変換される。

Z-score normalization は、標準化（Standardization）と呼ばれるのが一般的である。標準化とは、データの平均値からの偏差（＝平均値を中心 0 にした場合の値、中心化した値）を標準偏差で割ることである。これにより、データの平均は 0、標準偏差（＝データのバラツキ具合）は 1 に変換される（※なお、標準偏差 1 を二乗した「分散」の値も 1 なので、分散 1 とも表現できる）。

9.1.1 正規化

正規化（Min-Max 法）は、特にデータの最小値と最大値の範囲が明確な場合に適した手法である。ただし、外れ値に敏感なため、大きい外れ値が存在する場合は標準化を使った方がよい。

正規化（Min-Max 法）の数式は、以下のように定義できる。統計学に寄せて「観測値（observed value）」と表記したが、「実測値」「実際の測定値」の他、さまざまな方法で収集したデータがこの対象となる。

$$x'_i = \frac{x_i - min(x)}{max(x) - min(x)} \qquad (i = 1 \cdots n)$$

$$= \frac{\text{観測値}_{i\text{番目}} - \text{最小値(全観測値)}}{\text{最大値(全観測値)} - \text{最小値(全観測値)}}$$

この定義は「i 番目の観測値（つまり 1 つの値のみ）を正規化する式であること」に注意してほしい。データ全体を正規化するには、1 番目から n 番目までの各データを一つ一つ、この式を使って変換（＝スケーリング）していく必要がある。

9.1.2 標準化[14]

標準化は、データの分布が正規分布（ガウス分布）に従っている場合に特に効果的な手法である。ただし、必ずしもデータの分布が正規分布でなくても使えるので、データの最小値と最大値の範囲が「不」明確な場合など、正規化（Min-Max 法）があまり適切でないと考えられる場合にも標準化を用いるとよい。このため、正規化よりも標準化の方が使いやすいケースが多い。

前提条件として、全データから平均値を計算する式は次のようになる。

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$$

$$= \frac{1}{\text{データ数}} \sum_{\text{i番目=1から}}^{\text{データ数まで総和}} \text{観測値}_{\text{i番目}}$$

また、前提条件として、全データの標準偏差を計算する式は次のようになる。

$$\sigma = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^2}$$

$$= \sqrt{\frac{1}{\text{データ数}} \sum_{\text{i番目=1から}}^{\text{データ数まで総和}} (\text{観測値}_{\text{i番目}} - \text{平均値})^2}$$

標準化の数式は、上記の平均値と標準偏差の式を使って以下のように定義できる。

$$x_i' = \frac{x_i - \bar{x}}{\sigma} \qquad (i = 1 \cdots n)$$

$$= \frac{\text{観測値}_{\text{i番目}} - \text{平均値}}{\text{標準偏差}}$$

先ほどの正規化と同様に、この定義は「i 番目の観測値（つまり 1 つの値のみ）を標準化する式であること」に注意してほしい。標準化することで平均が 0 で分散が 1 となる。

9.1.3 batch Normalization を活性化関数の前と後ろのどっちに挿入

1. 活性化関数の前に BN を挿入

メリット

- 活性化関数への入力を正規化するため、勾配消失/爆発問題を緩和

理由

活性化関数に入る前にデータを正規化することで、各ニューロンの入力分布を安定させる
特に、ReLU などの非線形活性化関数の効果を最大化できる

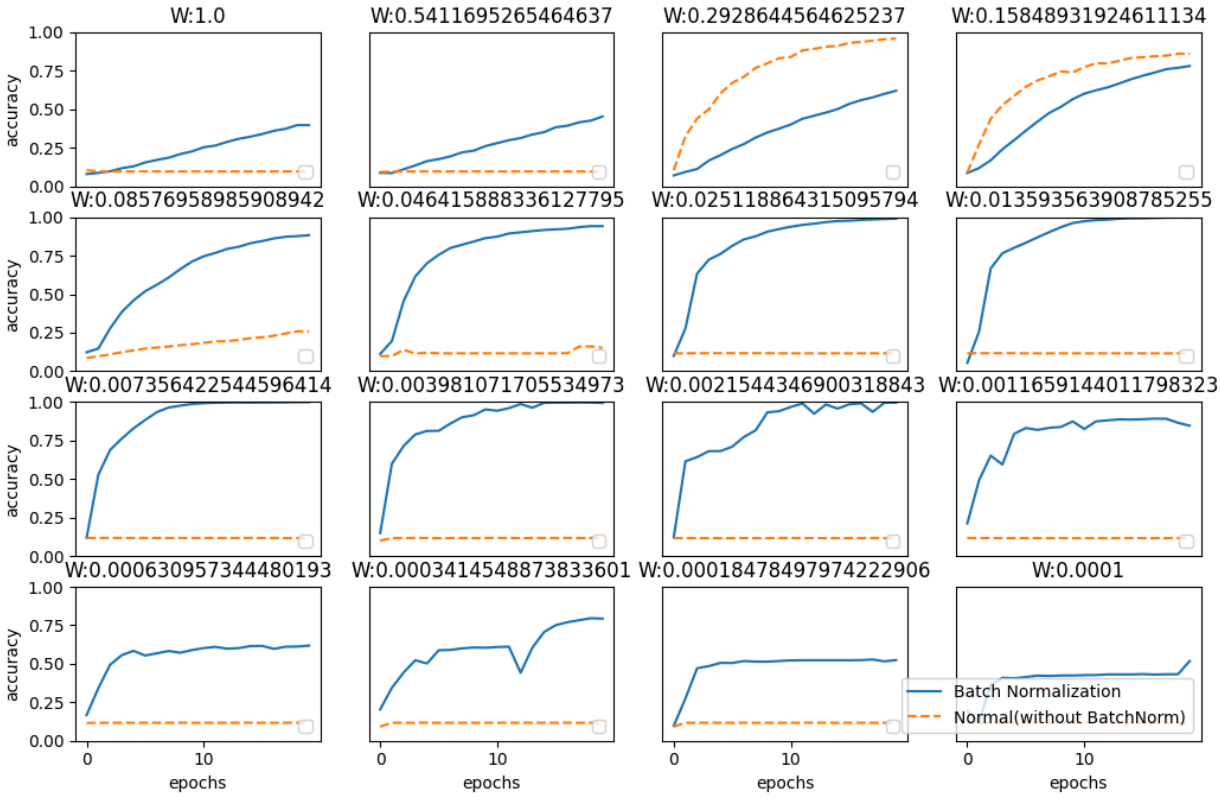
2. 活性化関数の後に BN を挿入

メリット

[14]https://manabitimes.jp/math/1043

• 活性化関数の出力を直接正規化するため、次の層への入力がより安定する可能性

理由
活性化関数によっては、出力の分布が偏る場合があり、それを正規化することで次の層の学習を安定させる



結果だけ載せておきます。

参考サイト：
https://qiita.com/Fumio-eisan/items/798351e4915e4ba396c2
https://ml4a.github.io/ml4a/jp/how_neural_networks_are_trained/